Algorithm 1 algpseudocode of l2c_ref_model

```
1:  while not end do
2:      //request arbitration
3:      if ! l2c_noc4_fifo.empty() then
4:          req_pkt ← l2c_noc4_fifo.pop();
5:      else if ! l2c_noc2_fifo.empty() & !l2c_noc3_fifo.nearfull() then
6:          req_pkt ← l2c_noc2_fifo.pop();
7:      else if ! req_buf.empty() & !l2c_noc1_fifo.empty() then
8:          req_pkt ← req_buf.pop();
9:      else
10:         req_pkt ← None;
11:     //request processing
12:     switch req_pkt.message_type do
13:         case LOAD_REQ
14:             do some actions in this case;
15:         case STORE_REQ
16:             do some actions in this case;
17:         case LOAD_FWD
18:             do some actions in this case;
19:         case STORE_REQ
20:             do some actions in this case;
21:         case DATA_EACK
22:             do some actions in this case;
23:         case NC_LOAD_REQ
24:             do some actions in this case;
25:         case NC_STORE_REQ
26:             do some actions in this case;
27:         case NC_LOAD_ACK, NC_STORE_ACK
28:             do some actions in this case;
29:         default
30:             do nothing
```

Algorithm 2 algpseudocode of l2c_ref_model

```
1: case LOAD_REQ
2:     if lru_valid(req_pkt.address) & lru_clean(req_pkt.address) then
3:         initialize evc_pkt;
4:         l2c_noc1_fifo.push(evc_pkt);
5:     else if lru_valid(req_pkt.address) & lru_dirty(req_pkt.address) then
6:         initialize wb_pkt;
7:         l2c_noc3_fifo.push(wb_pkt);
8:         initialize wbgurad_pkt;
9:         l2c_noc1_fifo.push(wbgurad_pkt);
10:    initialize load_pkt;
11:    l2c_noc1_fifo.push(load_pkt);
```

Algorithm 3 algpseudocode of mcu_ref_model

```
1: while not end do
2:     if !memi_noc4_fifo.empty() then
3:         if req_pkt.message_type == LOAD_MEM & !memo_noc4_fifo.nearfull() then
4:             req_pkt = memi_noc4_fifo.pop();
5:             data ← read_mem(req_pkt.address);
6:             initialize load_mem_ack;
7:             memo_noc4_fifo.push(load_mem_ack);
8:         else if req_pkt.message_type == STORE_MEM then
9:             req_pkt = memi_noc4_fifo.pop();
10:            write_mem(req_pkt.address, req_pkt.data);
```

Algorithm 4 Buffering Aware Spike Removal

```
1: function AverageProbeStream(spike_begin, spike_end)
2:     sum_sendgap ← 0, sum_recvgap ← 0
3:     if spike_begin < spike_end then
4:         for i = spike_begin → spike_end do
5:             sum_sendgap+ = send_gap[i]
6:             sum_recvgap+ = recv_gap[i]
7:         for i = spike_begin → spike_end do
8:             send_gap[i] = sum_sendgap ÷ (spike_end-spike_begin+1)
9:             recv_gap[i] = sum_recvgap ÷ (spike_end-spike_begin+1)
10: function SpikeRemoval
11:     i ← 0, spike_state ← NONE
12:     if recv_gap[0] > recv_gap[1] + SPIKE_DOWN then
13:         spike_begin ← 0
14:         spike_max ← recv_gap[0]
15:         spike_state ← SPIKE_VALID
16:         i ← 1
17:     for i → recv_gap.size()-1 do
18:         switch spike_state do
19:             case NONE
20:                 if recv_gap[i] + SPIKE_UP < recv_gap[i+1] then
21:                     spike_end ← i
22:                     AverageProbeStream(spike_begin,spike_end)
23:                     spike_state ← SPIKE_PENDING
24:                     spike_begin ← i+1
25:                     spike_max ← recv_gap[spike_begin]
26:                 break
27:             case SPIKE_PENDING
28:                 spike_max = max{spike_max, recv_gap[i]}
29:                 if recv_gap[i] + SPIKE_DOWN < spike_max then
30:                     spike_state ← SPIKE_VALID
31:                 else
                                    4
32:                     break
33:             case SPIKE_VALID
34:                 if recv_gap[i] + SPIKE_UP < recv_gap[i+1] then
35:                     spike_end ← i
36:                     spike_state ← SPIKE_PENDING
37:                     spike_max ← recv_gap[i+1]
38:                 else
39:                     if recv_gap[i]=recv_gap.back() then
```